Advances in Engineering Software 101 (2016) 37–49

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Synchronous and asynchronous HEVC parallel encoder versions based on a GOP approach

H. Migallón[a,*], J.L. Hernández-Losada[b], G. Cebrián-Márquez[b], P. Piñol[a], J.L. Martínez[b], O. López-Granado[a], M.P. Malumbres[a]

[a] Department of Physics and Computer Architecture, University Miguel Hernández, Elche, E-03202 Alicante, Spain
[b] Albacete Research Institute of Informatics, Universidad de Castilla-La Mancha, E-02071 Albacete, Spain

ARTICLE INFO

Article history:
Available online 19 February 2016

Keywords:
Parallel algorithms
Video coding
HEVC
Multicore
Performance
GOP-based algorithms
Synchronous algorithms
Asynchronous algorithms

ABSTRACT

In this paper, we focus on applying parallel processing techniques to HEVC encoder in order to significantly reduce the computational power requirements without disturbing its coding efficiency. So, we propose several, synchronous and asynchronous, parallelization approaches working at a coarse grain parallelization level, based on the Group Of Pictures (GOP), which we call GOP-based level. GOP-based approaches encode simultaneously several groups of consecutive frames. Depending on how these GOPs are conformed and distributed it is critical to obtain good parallel performance . The results show that near ideal efficiencies are obtained using up to 10 cores. Furthermore, when the computational load is unbalanced, the asynchronous versions outperform the synchronous ones. The parallel algorithms developed in this work support all standard coding modes proposed by the reference software.

1. Introduction

Recently, the Joint Collaborative Team on Video Coding (JCT-VC) co-established by ISO/IEC MPEG (Motion Experts Group) and ITU-T VCEG (Video Coding Experts Group), has standardized the next-generation video coding technology called High Efficiency Video Coding (HEVC) [1]. This new standard will replace the current H.264/AVC (Advanced Video Coding) [2] standard in order to deal with nowadays and future multimedia market trends, since 4K definition video content is a nowadays fact and 8K definition video will not take too long to become a reality. Even more, the new standard supports high quality color depth at 8 and 10 bits. HEVC greatly improved the coding efficiency over its predecessor (H.264/AVC) by a factor of almost twice while maintaining an equivalent visual quality [3].

Regarding complexity, in [4], Bossen et al. studied the complexity aspects of HEVC encoding and decoding software. This study concludes that the encoding process is much more challenging than the decoding process, e.g., encoding one second of a 1080p60 HD (High Definition) video with the reference software encoder can take longer than one hour when running in an off-the-shelf desktop computer. Therefore, HEVC encoder optimization will be a hot research topic in years to come.

Several works about complexity analysis and parallelization strategies for the emerging HEVC standard can be found in the literature [4–6]. Most of parallelization proposals are focused in the decoding side, looking for the most appropriate parallel optimizations at the decoder that provide real-time decoding of High-Definition (HD) and Ultra-High-Definition (UHD) video contents. In [7,8] the authors present a variation of Wavefront Parallel Processing (WPP) called Overlapped Wavefront (OWF) for the HEVC decoder in which the executions over consecutive pictures are overlapped. In a multi-threaded approach of the HEVC decoder, a picture is decoded by several threads at the same time, being each thread in charge of decoding different Coding Tree Block (CTB) rows. In these works, authors claim that a single thread may continue processing the next picture when it finishes the current one, without waiting for the other threads. These variations allow a better parallel processing efficiency, reducing the overall decoding time. Recently, in [9] the authors mixed tiles, WPP and SIMD (Single-Instruction Multiple-Data instruction set extension to the x86 architecture) instructions to develop a real-time HEVC decoder.

At the moment, only a few works focused on the HEVC encoder have been reported. In [10] authors propose a fine-grain parallel optimization of the HEVC motion estimation module that performs at the same time the motion prediction of all Prediction Units (PUs) available at one Coding Unit (CU). In [11,12] authors propose a real-time motion estimation block over focusing on the optimization of motion estimation algorithms using an FPGA-based low cost embedded system with a combination of synchronous

* Corresponding author. Tel.: +34 966658390; fax: +34 966658814.
E-mail address: hmigallon@umh.es (H. Migallón).

http://dx.doi.org/10.1016/j.advengsoft.2016.01.020
0965-9978/

dynamic random access memory (SDRAM) with on-chip memory of software-based Nios II processors. Through the optimizations of memory access in this platform, time savings of up 53% were achieved in the motion estimation module. In [13] authors propose a parallelization at the intra prediction module that consists on removing data dependencies among subblocks of a CU, obtaining interesting speed-up results with a negligible loss in coding performance. Other recent works are focused on changes in the scanning order. For example, in [14] the authors propose a CU scanning order based on a diamond search obtaining a good scheme for massive parallel processing. Also in [15] the authors propose to change the HEVC deblocking filter processing order obtaining time savings of 37.93% over many-core architectures.

In this paper, we will focus on applying parallel processing techniques to the HEVC encoder in order to significantly reduce the computational power requirements without disturbing the coding efficiency. Instead of focusing the optimization on one specific module of the HEVC encoder, as other proposals do, our proposals use OpenMP programming paradigm working at a coarse grain parallelization level which we call GOP-based level. GOP-based approaches encode simultaneously several Group Of Pictures (GOP). Depending on how these GOPs are conformed and distributed it is critical to obtain good parallel performance, taking also into account the level of coding efficiency degradation. This paper is based upon Migallón et al. [16], including more results and additional research such (a) new asynchronous parallel versions, and (b) a comparison between synchronous and asynchronous parallel versions in terms of computational complexity (coding time) and speed-up.

The remainder of this paper is organized as follows, in Section 2 an overview of the available profiles and parallel strategies in HEVC are presented. Sections 3 and 4 describe the GOP-based parallel alternatives proposed for both synchronous and asynchronous architectures , while in Section 5 a comparison between the proposed parallel approaches is presented. Finally, in Section 6 some conclusions are drawn.

2. HEVC coding modes and parallel strategies

HEVC follows a hybrid video coding scheme consisting on a sequence of three main steps. First, the spatial or temporal redundancy is exploited to make a prediction of a frame region and, in this way, only the residuum of the prediction and some side information will be encoded. In the second step, the residuum is transformed into the frequency domain and the resulting coefficients are quantized (lossy compression). Depending on the quantization step, we will achieve a higher or lower compression ratio in the bit stream, and the reconstructed video sequence will exhibit a higher or lower visual quality. In the third step, entropy coding is applied to the quantized coefficients and the side information in order to further compress the bit stream.

In the encoding process, each frame is divided into small square regions called Coding Units (CU). Spatial redundancy is exploited by obtaining a prediction for a CU (or CU subpartitions) using the nearby pixels in the same frame. Temporal redundancy is exploited by searching for similar CUs (or CU subpartitions) in previous or past frames in order to obtain a prediction.

In our tests we have used the available coding modes of the reference software package [17]: All Intra (AI), Random Access (RA), Low-Delay B (LB), and Low-Delay P (LP). Each mode has different characteristics and can be used in different situations, depending on the requirements of each application.

In All Intra mode every frame is coded as an I-frame, i.e., it is encoded without any motion estimation/compensation. So, each frame is independent from the other frames in the sequence. This mode gets lower compression rates compared to the other 3 modes, but the encoding process is faster. Fig. 1 shows the AI mode coding structure.

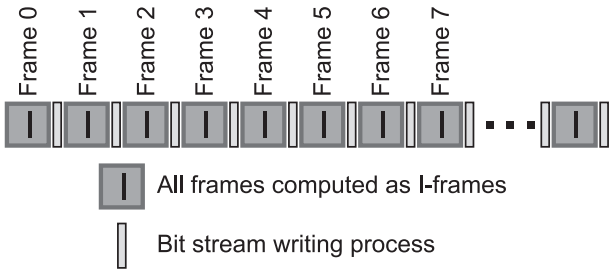


Fig. 1. All Intra (AI) mode sequential structure.

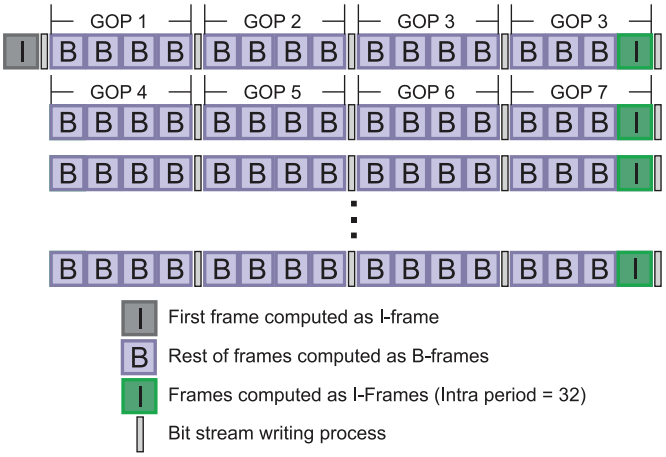


Fig. 2. Random Access (RA) mode sequential structure.

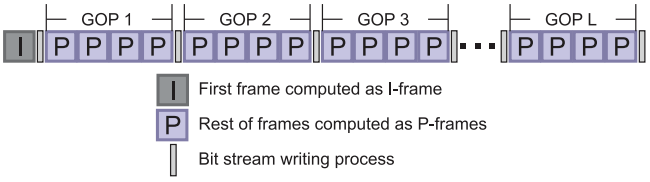


Fig. 3. Low delay P (LP) mode sequential structure.

Random Access mode combines I-frames and B-frames. In this mode, reference frames used to perform the motion estimation process can be located earlier or later than the frame we are currently encoding. So, encoding (and decoding) order is not the same as rendering order. In the HEVC standard, the RA mode uses a GOP size of 8 frames. To allow navigating through the coded sequence (pointing to a certain video sequence frame) or to allow trick modes like fast forward, an I-frame is inserted periodically. The intra refresh period must be a multiple of GOP size. Fig. 2 shows the coding structure for the RA mode when the intra refresh period is equal to 32 frames. The bit stream updating is performed after each GOP computation.

Low-Delay modes (LP and LB) encode each frame in rendering order. First an I-frame is inserted in the coded bit stream and then only P-frames (or B-frames) are used for the rest of the sequence, with a GOP size equal to 4. All the reference pictures are located earlier than the current frame. These two modes achieve better compression performance than AI mode and do not suffer from the delay that RA mode introduces. The coding structure of the Low Delay P (LP) and Low Delay B (LB) modes shown in Figs. 3 and 4, respectively, are very similar.

Once we have presented all available coding modes in HEVC, we will show the different parallel techniques that can be applied.

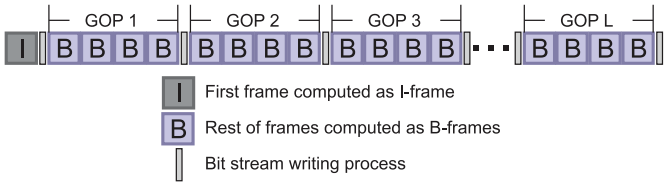


Fig. 4. Low delay (LB) mode sequential structure.

High-level parallel strategies may be classified in a hierarchical scheme depending on the desired parallel grain size. This classification should be carefully applied, taking into account the available parallel hardware resources in order to perform the most adequate and efficient implementation. So, we define from coarser to finer grain the following parallelism levels: GOPs, slices, tiles, and Wavefront Parallel Processing (WPP). When designing an HEVC parallel version we first analyze the available hardware where the parallel encoder will run, in order to determine which parallel scheme is the most appropriate.

The coarsest parallelization level, GOP-based, is based on dividing the whole video sequence in GOPs in such a way that the processing of each GOP is completely independent from the processing of the other GOPs. In general, this approach can obtain good parallel efficiency on both shared memory and distributed memory architectures. However, depending on the way we define the GOPs structure and remove the inter-GOP dependencies, the coding performance may be affected.

Other parallelization levels (tiles, slices and WPP) work at a frame fragment scale and are not suitable for a distributed memory platforms. We will perform all our tests using diverse GOP-based parallelization configurations.

Current reference software does not directly support most of the high-level parallelism approaches mainly due to its implementation design. In the next section we will present the GOP-based parallelization approaches tested. They may be implemented in distributed, shared or hybrid memory architectures.

3. Synchronous GOP-based parallel algorithms

We have developed several strategies, described below, but in all of them one GOP is computed by one process. Firstly, we propose synchronous algorithms, in which the synchronization processes are located after the encoding process of a GOP, and the reference pictures are not shared. The main goals of these mild restrictions are, on the one hand, to fill the bit stream as the information is available, and on the other hand, to be able to extend the work to distributed memory platforms without drastically increasing the amount of information that should be transmitted.

Note that the AI mode differs from the rest of modes on both the GOP size (equal to 1 for the AI mode and greater than 1 for the rest of modes), and in that all frames are computed as I-frames, which means that no reference frames are used. Therefore the AI parallel algorithm does not differentiate from the sequential algorithm regarding video quality and bit rate.

We have developed five synchronous parallel approaches. All of them support the Low Delay B (LB), Low Delay P (LP) and Random Access (RA) modes, except one of them that only supports All Intra (AI) mode. The first four approaches we have developed are:

- Option S-I (LB, LP and RA): All processes encode the first I-frame and include it in the reference picture list. After that, each GOP is allocated to every process in a round-robin fashion. So, processes will encode isolated GOPs that link to the reference picture list.
- Option S-II (LB, LP and RA): The video sequence is divided in as many chunks as the number of available parallel processes. Before processing the corresponding block of adjacent GOPs, each process encodes the first GOP, which is composed by the first I-frame of the video sequence.
- Option S-III (LB, LP and RA): In this approach the video sequence is divided in as many chunks as the number of processes, in a similar way than in Option S-II, but in this case, each process starts encoding one I-frame at the beginning of its block of adjacent GOPs, altering the frame pattern of the corresponding encoding mode.
- Option S-IV (AI): As in Option S-I each GOP is allocated to each process following a round-robin scheme. However, all GOPs consist of a single I-frame.

Fig. 5 shows the parallel distribution performed when Option S-I is used. As we have said, a synchronization process is located after each GOP is encoded. All processes compute the first GOP, i.e., one I-frame, but obviously only the root process (P0) writes data into the bit stream. After that, each process encodes a GOP of 4 frames (or 8 frames for RA mode). The GOP assigned to each process depends on the rank of the parallel process, because GOPs are sequentially assigned to each process.

As each process will have its own working buffer in order to store the reference picture list, the real pattern of the reference pictures used changes for parallel and sequential algorithms and it also changes depending on the number of processes used in the parallel algorithm. Note that each process builds its own buffer with the reference picture list, numbering the frames according to the order they are stored in the local buffer. For instance, in a sequential process, the second frame of a GOP uses frames *−1, −2, −6* and *−10* as reference pictures (*−1* means the previous frame, and so on). Considering a GOP size equal to 4 (as in LB and LP modes), frame *−2* points to the last frame of the previous GOP (the frame two positions before the current frame in the original video sequence). In a parallel process, as we assign isolated GOPs to each process, the previous GOP is not the previous adjacent GOP in the original video sequence and therefore frame *−2* will not point to

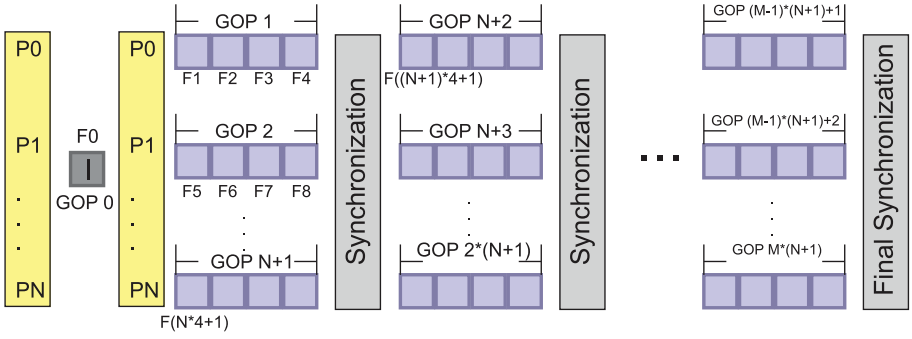


Fig. 5. Option S-I: parallel distribution.

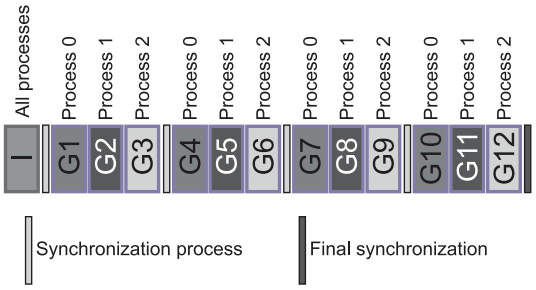


Fig. 6. Option S-I: example of parallel distribution.

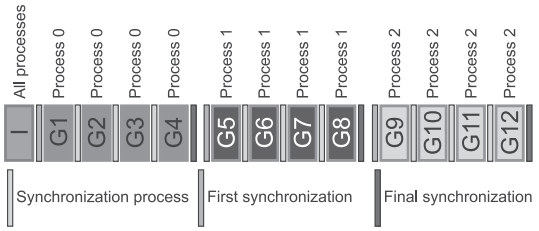


Fig. 8. Option S-II: example of parallel distribution.

the frame two positions before the current frame. If, for instance, the number of processes is 6, then the previous GOP for this process will be located in the video sequence 6 GOPs away from the current GOP. So for the second frame of a GOP, the reference picture *–2* will point to frame *–22 (–2–(6–1) × 4 = –22)* in the original video sequence. We can conclude that both parallel and sequential algorithms will produce different bit streams. Later, we will analyze the impact of this fact in terms of PSNR and bit rate, and we will propose several parallel strategies in order to minimize this issue.

In Fig. 6, in order to clarify the Option S-I parallel distribution, we present an example of a video sequence with 49 frames (the first I-frame and 12 GOPs of 4 frames) is encoded with 3 parallel processes. As we can see, all processes encode the first I-frame and after that process 0 encodes GOPs 1, 4, 7, and 10, while process 1 encodes GOPs 2, 5, 8, and 11 and finally, process 2 encodes GOPs 3, 6, 9, and 12. As we have said, we perform a synchronization process after a GOP is encoded by each process.

The parallel distribution performed in Option S-II is represented in Fig. 7. In this proposal we still perform a synchronization process after each GOP is encoded. All processes compute the first frame as an I-frame, that is, the first GOP, but obviously again only the root process (P0) writes data into the bit stream. This first frame is included in the reference picture list of all processes, but in this case the reference pictures are not so much disturbed, because each process works with a group of adjacent GOPs. In the previous example, the pattern is only altered for the first three GOPs. From this point onward all reference pictures needed are available in the private working buffer of each process.

Fig. 8 presents a similar example for Option S-II parallel distribution. All processes encode the first I-frame and after that, each process encodes adjacent GOPs. Process 0 encodes GOPs 1, 2, 3, and 4, while process 1 encodes GOPs 5, 6, 7, and 8 and process 2 encodes GOPs 9, 10, 11, and 12. We still perform a synchronization process when all processes finish the computation of one GOP.

The parallel distribution of Option S-III, shown in Fig. 9, presents a parallel structure similar to Option S-II. We still assign a block of adjacent GOPs to each process, but we change the structure of frames indicated by LB, LP and RA modes, since in this case each process does not encode the first frame of the video sequence as an I-frame. Instead of that, each process encodes the first frame of its block of adjacent GOPs as an I-frame.

In the example shown in Fig. 10, for the Option S-III algorithm, we consider that 3 parallel processes encode 51 frames. In this case, each process encodes 17 frames. The video sequence is partitioned in 3 chunks each one containing 17 frames. Each process encodes the first frame of its chunk as an I-frame, and after that, each process encodes four adjacent GOPs.

As previously mentioned, Option S-IV parallel strategy is the same as Option S-I, but working only with the AI mode. Fig. 11 shows the parallel distribution for Option S-IV. Note that a GOP always consists of one I-frame, and moreover these I-frames are IDRs (Instantaneous Decoder Refresh), therefore there are no differences between the parallel and the sequential execution, because this mode does not use the reference picture list. In this case, as in Option S-I, in the synchronization processes after the encoding of each GOP, the bit stream can be updated with data provided by all processes, while in Option S-II and Option S-III the bit stream can only be updated with data provided by the root process. Consequently, only at the end of the video encoding we can update the bit stream with the data provided by the rest of processes.

As we have previously said, Option S-IV is designed for the AI mode, where each I-frame is one GOP. Therefore, in Fig. 12 we present an example encoding 12 GOPs (i.e. 12 frames) using 3 parallel processes. The synchronization process is performed after all processes have encoded one frame (i.e one GOP).

Finally, we have developed another strategy, named Option S-V, with two main goals, (a) all processes are able to write data in the bit stream more frequently than Options S-II and S-III, and (b) to minimize changing the frame pattern of the standard LB, LP and RA modes, in order to reduce PSNR and bit rate differences with respect to the sequential algorithm. The proposed Option S-V strategy combines features of the other presented proposals. Firstly, all processes encode the first GOP, composed by a single I-frame, introducing it in each local buffer which stores the reference picture list. In the same way as in Option S-I and Option S-II, only

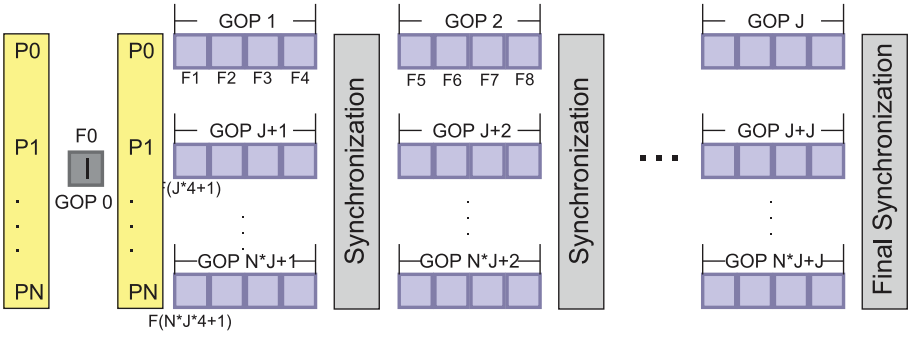


Fig. 7. Option S-II: parallel distribution.

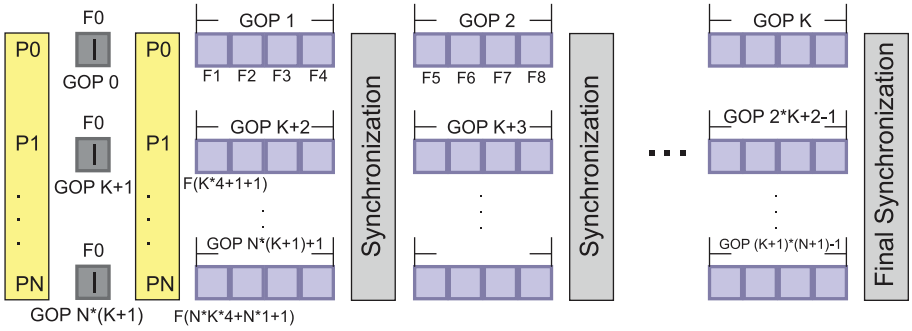


Fig. 9. Option S-III: parallel distribution.

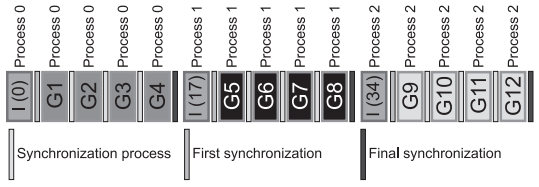


Fig. 10. Option S-III: example of parallel distribution.

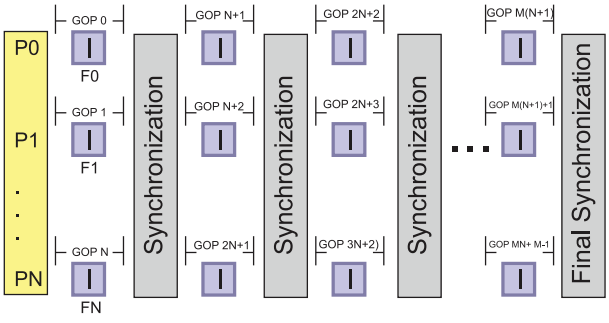


Fig. 11. Option S-IV: parallel distribution.

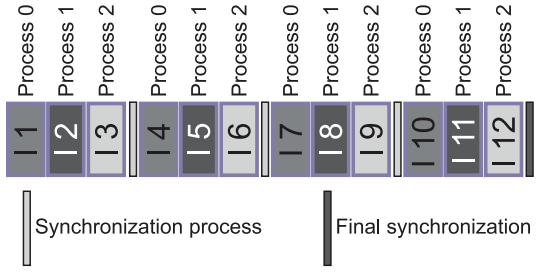


Fig. 12. Option S-IV: example of parallel distribution.

the root process updates the bit stream. After that, a fixed number of adjacent GOPs (named GOP_BLOCK) are assigned to each process depending on their parallel rank. Obviously the size of the GOP_BLOCK must be the same for all processes. Fig. 13 shows

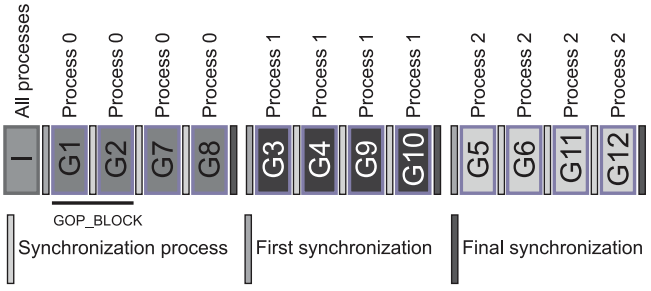


Fig. 14. Option S-V: example of parallel distribution.

Option S-V parallel structure when the size of the GOP_BLOCK is equal to 3. Note that the root process can update the bit stream after each GOP computation, while the rest of processes can update the bit stream after the GOP_BLOCK computation. We want to remark that as we increase the GOP_BLOCK size, the disturbance of the reference picture list decreases.

For the Option S-V example shown in Fig. 14, we encode 49 frames using 3 parallel processes. The GOP_BLOCK is equal to 2, thus process 0 firstly encodes GOPs number 1, 2, whereas processes 1 and 2 encode GOPs number 3, 4, and 5, 6, respectively. After that, process 0 encodes GOPs number 7, 8, whereas processes 1 and 2 encode GOPs 9, 10, and 11, 12, respectively. We still perform a synchronization process when all processes finish the computation of one GOP.

Remark that Figs. 5, 7, 9 and 13 show parallel distributions considering the GOP size is equal to 4, like for LB and LP modes, but for RA mode the GOP size is equal to 8. Regarding Option S-V algorithm the GOP_BLOCK size sets the number of GOPs. Thus, in the example shown in Fig. 13, the number of frames included in one GOP_BLOCK is equal to 12 (three GOPs of four frames) for LB and LP modes, while for RA mode it is equal to 24 (three GOPs of eight frames).

3.1. Synchronous algorithms evaluation

In this subsection we analyze the synchronous parallel algorithms described previously in terms of parallel performance, PSNR

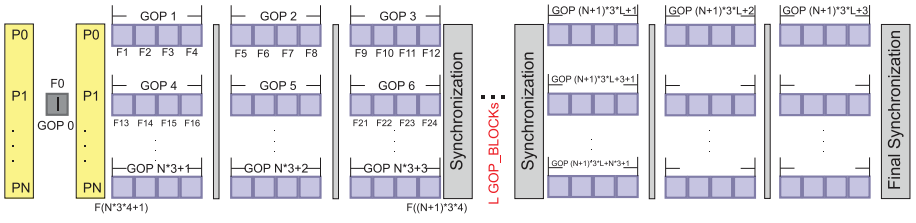
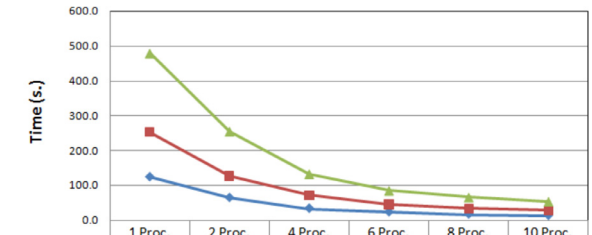
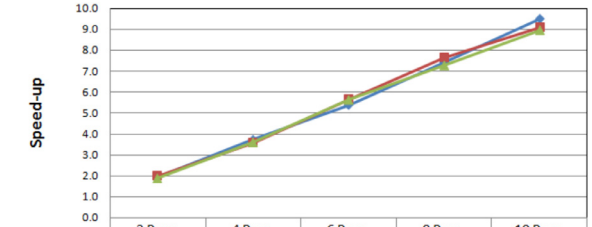


Fig. 13. Option S-V: parallel distribution.



	1 Proc.	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
120 Frames	124.5	64.4	33.4	23.1	16.8	13.1
240 Frames	252.4	126.6	70.4	44.6	33.1	27.8
480 Frames	478.3	254.4	132.6	85.0	65.7	53.4

(a) Time (s.)



	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
120 Frames	1.9	3.7	5.4	7.4	9.5
240 Frames	2.0	3.6	5.7	7.6	9.1
480 Frames	1.9	3.6	5.6	7.3	9.0

(b) Speed-up

Fig. 15. Option S-IV parallel algorithms when computing 120, 240 and 480 frames.

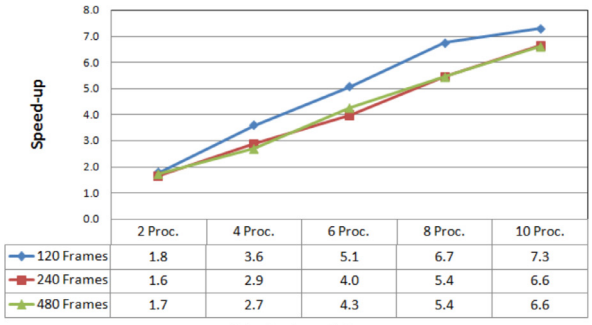
and bit rate. As the experiments reported were obtained on a shared memory platform, we have used the OpenMP [18] programming paradigm. In particular the multicore platform used is a HP Proliant SL390 G7 with two Intel Xeon X5660, each CPU with six cores at 2.8 GHz, therefore the experiments reported use up to 10 processes. The testing video sequence used is *BQSquare_416x240_60.yuv*, and disposes of 600 frames at 60*Hz* with a frame size equal to 416*x*240 pixels. We have run the parallel algorithms encoding 120, 240 and 480 frames for All Intra (AI), Low Delay B (LB) and Low Delay P (LP) modes, and encoding 256 and 512 frames for Random Access (RA) mode. The Group of Pictures (GOP) size for LB and LP modes is equal to 4 while for RA mode is equal to 8. Also, LB and LP modes compute only the first frame as I-frame, while RA mode inserts one I-frame (IDR type) every 32 frames in our experiments. In the AI mode all frames are I-frames and one GOP consists on one I-frame. In all cases the value of quantization parameter (QP) is equal to 32. Finally, version 10.0 of the HEVC reference software has been used and the experiments reported were performed using the Main profile (i.e. bit depth of 8 bits)

In Fig. 15, we present computational results for Option S-IV parallel algorithm. Fig. 15(a) shows the computational times when encoding 120, 240 and 480 frames, and Fig. 15(b) shows the corresponding speed-up. Note that Option S-IV is the only algorithm related to the AI mode, in which there are no dependencies between frames. This parallel algorithm offers good time reductions when increasing the number of processes, achieving efficiencies close to the ideal ones. Remark that in Option S-IV the reference sequential execution (i.e using 1 process) obtains the same bit rate and PSNR results than the parallel execution.

In Table 1 we present the computational times for Option S-I, Option S-II and Option S-III parallel algorithms using the LB encoding mode. Note that when using just 1 process, all the proposed algorithms show similar timings than the ones obtained by the sequential version. Fig. 16 shows the speed-ups associated to the results shown in Table 1. This figure confirms the good behavior of the proposed parallel algorithms, obtaining good speed-ups in

Table 1
Computational times for the parallel algorithms for the LB mode when computing 120, 240 and 480 frames.

Processors	#Frames	S-I	S-II	S-III
1	120	437.8	438.4	426.9
	240	884.8	887.7	909.9
	480	1807.2	1811.7	1803.6
2	120	247.2	231.7	**223.8**
	240	541.4	479.8	**456.5**
	480	1054.9	945.0	**906.9**
4	120	122.2	**112.3**	114.8
	240	308.4	248.4	**237.3**
	480	674.1	489.7	**473.7**
6	120	86.4	79.4	**76.2**
	240	223.0	172.3	**158.3**
	480	425.1	327.1	**317.3**
8	120	65.0	59.7	**57.3**
	240	162.7	**121.0**	125.9
	480	332.2	250.8	**238.0**
10	120	60.1	52.7	**50.1**
	240	133.3	106.8	**98.0**
	480	273.7	203.2	**197.5**

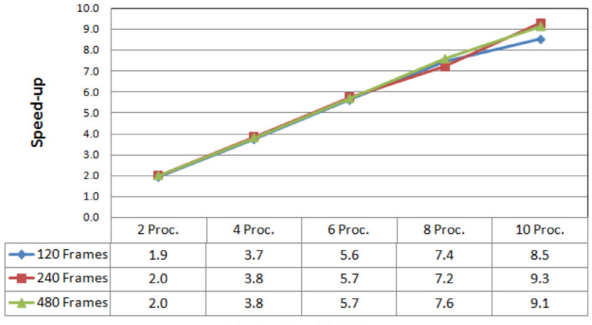


	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
120 Frames	1.8	3.6	5.1	6.7	7.3
240 Frames	1.6	2.9	4.0	5.4	6.6
480 Frames	1.7	2.7	4.3	5.4	6.6

(a) Option S-I.

	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
120 Frames	1.9	3.9	5.5	7.3	8.3
240 Frames	1.9	3.6	5.2	7.3	8.3
480 Frames	1.9	3.7	5.5	7.2	8.9

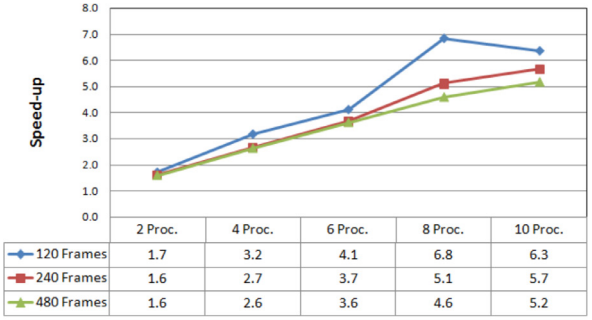
(b) Option S-II.



	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
120 Frames	1.9	3.7	5.6	7.4	8.5
240 Frames	2.0	3.8	5.7	7.2	9.3
480 Frames	2.0	3.8	5.7	7.6	9.1

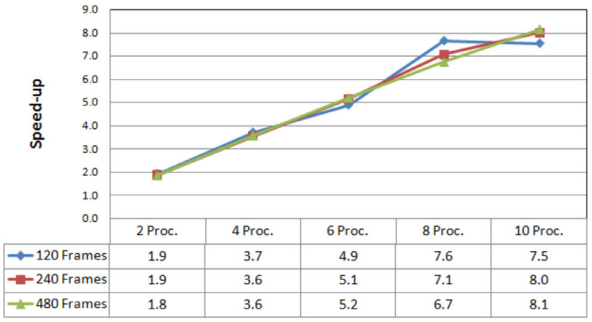
(c) Option S-III.

Fig. 16. Speed-up for the parallel algorithms for the LB mode when computing 120, 240 and 480 frames.



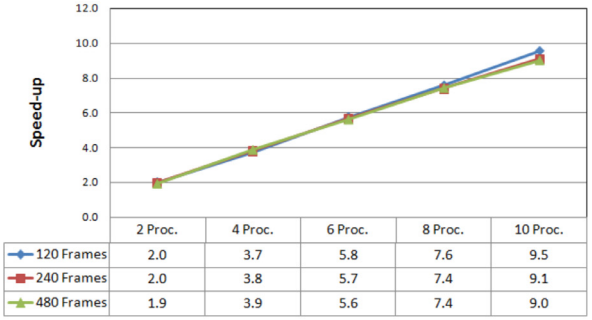
	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
120 Frames	1.7	3.2	4.1	6.8	6.3
240 Frames	1.6	2.7	3.7	5.1	5.7
480 Frames	1.6	2.6	3.6	4.6	5.2

(a) Option S-I.



	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
120 Frames	1.9	3.7	4.9	7.6	7.5
240 Frames	1.9	3.6	5.1	7.1	8.0
480 Frames	1.8	3.6	5.2	6.7	8.1

(b) Option S-II.



	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
120 Frames	2.0	3.7	5.8	7.6	9.5
240 Frames	2.0	3.8	5.7	7.4	9.1
480 Frames	1.9	3.9	5.6	7.4	9.0

(c) Option S-III.

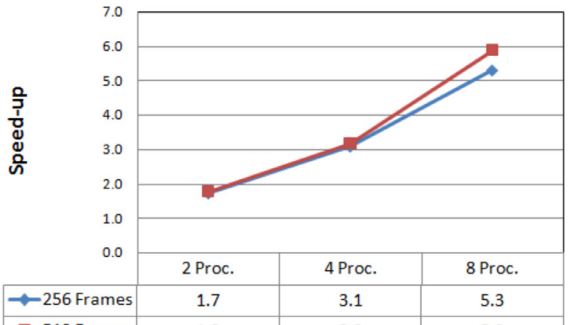
Fig. 17. Speed-up for the parallel algorithms for the LP mode when computing 120, 240 and 480 frames.

all cases. However, the results obtained using both Option S-II and Option S-III are significantly better than those obtained by Option S-I. Note that the reference picture list used in Option S-I does not include adjacent GOPs.

Fig. 17 shows the speed-ups when parallel algorithms encode the video sequence using the LP encoding mode. It should be noted that the results shown are better than those obtained with the LB mode. In particular, when using Option S-III parallel algorithm, the speed-up results are as good as those obtained using the S-IV parallel algorithm.

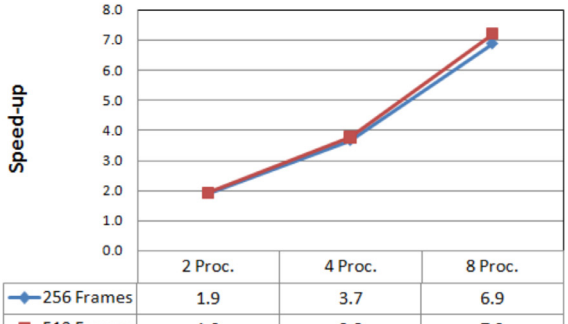
Taking into account all previous computational results, we can conclude that Option S-II and Option S-III obtain better performance than Option S-I, when LB or LP modes are used. In order to analyze the computational behavior when the RA mode is used we test our parallel proposals encoding 256 and 512 frames. We have selected this number of frames because the RA mode works with a GOP size equal to 8 and it inserts an I-frame every 32 frames. Looking at the results shown in Fig. 18, similar conclusions are obtained for RA mode.

In order to confirm that the results can be extended to other video sequences, in Fig. 19 we present the speed-up obtained by Option S-III encoding 240 frames in LB mode for the following four videos sequences: *BasketballPass_416x240_50.yuv*



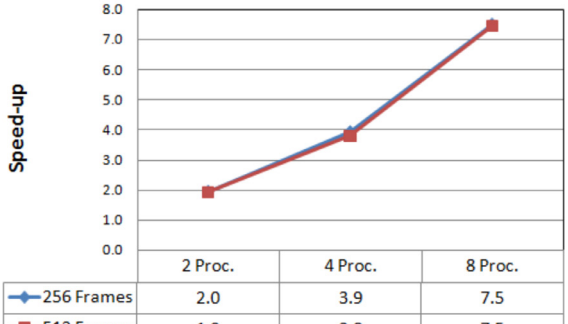
	2 Proc.	4 Proc.	8 Proc.
256 Frames	1.7	3.1	5.3
512 Frames	1.8	3.2	5.9

(a) Option S-I.



	2 Proc.	4 Proc.	8 Proc.
256 Frames	1.9	3.7	6.9
512 Frames	1.9	3.8	7.2

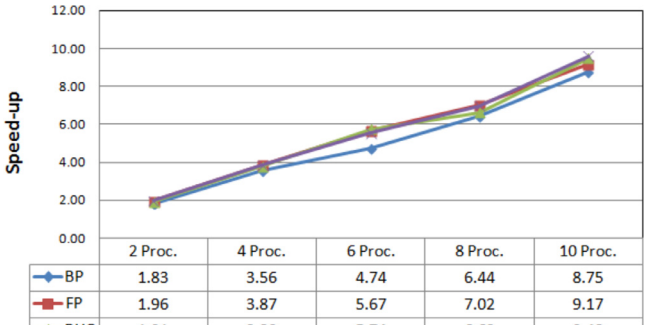
(b) Option S-II.



	2 Proc.	4 Proc.	8 Proc.
256 Frames	2.0	3.9	7.5
512 Frames	1.9	3.8	7.5

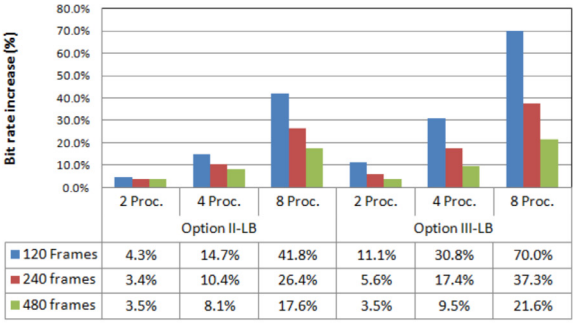
(c) Option S-III.

Fig. 18. Speed-up for the parallel algorithms when encoding in RA mode computing 256 and 512 frames.

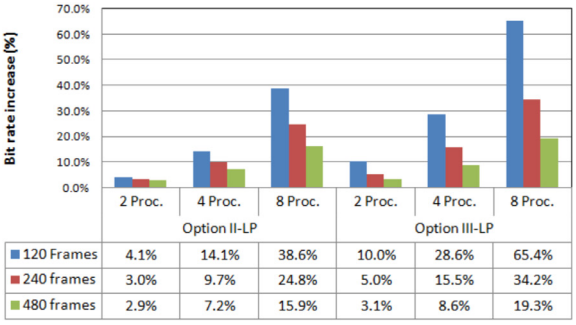


	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
BP	1.83	3.56	4.74	6.44	8.75
FP	1.96	3.87	5.67	7.02	9.17
RHC	1.91	3.80	5.74	6.63	9.48
BQT	1.96	3.85	5.54	6.97	9.59

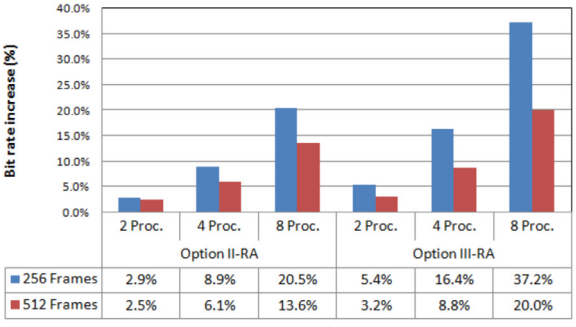
Fig. 19. Speed-up for various video sequences when encoding in LB mode computing 240 frames using Option S-III algorithm.



(a) LB mode.



(b) LP mode.



(c) RA mode.

Fig. 20. Percentage of bit rate increase for the parallel algorithms.

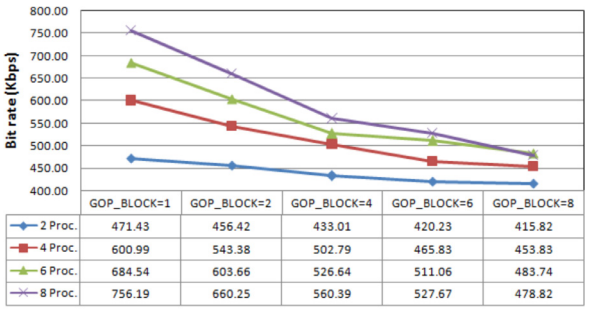
(BP), *FourPeople_1280x720_60.yuv* (FP), *RaceHorses _832x480_30.yuv* (RHC) and *BQTerrace_1920x1080_60.yuv* (BQ). As it can be seen in Fig. 19, the behavior for all sequences is similar. It is important to remark that we have used video sequences with different frame resolutions and frame rates.

As previously mentioned, the parallel versions do not provide the same results than the ones produced by the sequential algorithm. So, in Fig. 20 we show how parallel versions increase the bit rate obtained with respect to the sequential version. It is important to remark that Fig. 20 shows results for options S-II and S-III, but not for Option S-IV, because in this case the parallel and the sequential versions exhibit the same bit rate. Furthermore, we can observe that the bit rate increase introduced by Option S-I algorithm is not acceptable. This algorithm drastically changes the structure of the reference pictures and as a consequence it causes the large bit rate increase. In all cases the bit rate increase becomes larger as the number of processes does.

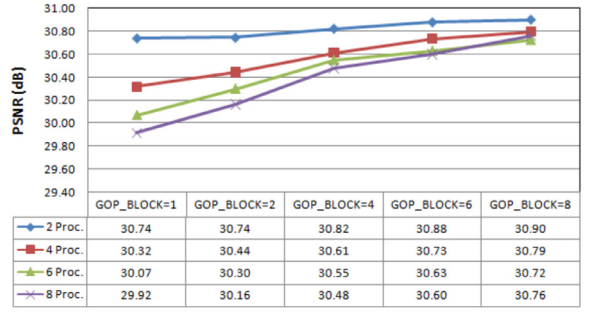
Table 2 shows the PSNR data, i.e., video quality measurement, for the parallel algorithms S-II and S-III. We can observe that the quality of the encoded video decreases when using Option S-II algorithm, although, in Fig. 20, we have shown that the bit rate increases. However, the bit rate increase introduced by the Option S-III algorithm, shown in Fig. 20, is compensated by a quality increase as it can be seen in Table 2.

Table 2
Luminance PSNRs (dB) for parallel algorithms.

Algorithms	1 Proc	2 Proc	4 Proc	8 Proc
Option S-II-LB-480 frames	31.28	31.24	31.19	31.04
Option S-III-LB-480 frames	31.28	31.29	31.32	31.37
Option S-II-LP-480 frames	31.15	31.11	31.05	30.94
Option S-III-LP-480 frames	31.15	31.16	31.18	31.21
Option S-II-RA-512 frames	31.92	31.90	31.87	31.80
Option S-III-RA-512 frames	31.92	31.93	31.94	31.95



(a) Bit rate.



(b) PSNR.

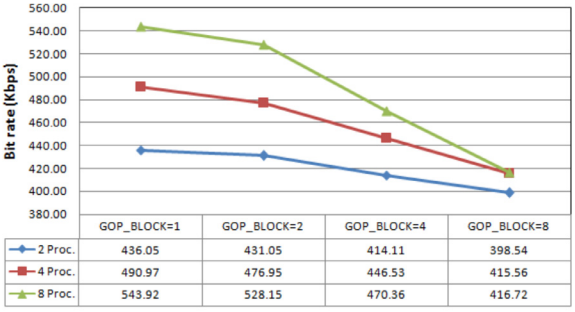
Fig. 21. Bit rate and PSNR for Option S-V algorithm varying the GOP_BLOCK size for LP mode.

In order to analyze Option S-V parallel algorithm, we will take into account a new parameter which modifies the parallel algorithm. As we saw in Fig. 13, in Option S-V parallel algorithm one block of consecutive GOPs is assigned to each process. The number of GOPs of each block, GOP_BLOCK, is the new parameter. Note that if the GOP_BLOCK is equal to 1, both Option S-V and Option S-I parallel algorithms are identical, while if the GOP_BLOCK size is equal to *Number of Frames to Encode / Number of Processes* Option S-V and Option S-II parallel algorithms are the same. Now, we will analyze Option S-V algorithm attending to bit rate and PSNR data. The results reported were obtained encoding 256 and 512 frames. As we encode 512 frames using 8 processors, each process computes 64 frames, i.e., 16 GOPs when LB or LP mode is used and 8 GOPs when RA mode is used. As expected, as we increase the GOP_BLOCK size the bit rate decreases, see Fig. 21(a) for LP mode and Fig. 22(a) for RA mode, and also the PSNR improves, see Figs. 21(b) and 22(b) for LP and RA modes, respectively.

After analyzing in detail the synchronous proposals, we can assess that good computational results are obtained without appreciably affecting the performance of the HEVC encoder.

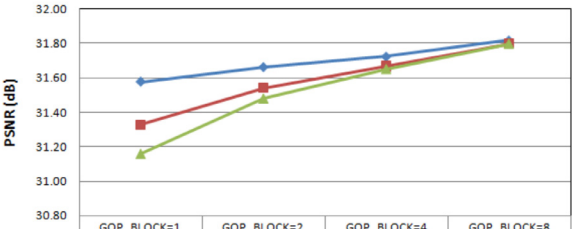
4. Asynchronous GOP-based parallel algorithms

The synchronization processes in the previous algorithms imply a fixed and structured Group of Pictures (GOP) computation, i.e.,



	GOP_BLOCK=1	GOP_BLOCK=2	GOP_BLOCK=4	GOP_BLOCK=8
2 Proc.	436.05	431.05	414.11	398.54
4 Proc.	490.97	476.95	446.53	415.56
8 Proc.	543.92	528.15	470.36	416.72

(a) Bit rate.



	GOP_BLOCK=1	GOP_BLOCK=2	GOP_BLOCK=4	GOP_BLOCK=8
2 Proc.	31.58	31.66	31.72	31.82
4 Proc.	31.33	31.54	31.67	31.80
8 Proc.	31.16	31.48	31.65	31.80

(b) PSNR.

Fig. 22. Bit rate and PSNR for Option S-V algorithm varying the GOP_BLOCK size for RA mode.

the number and the order of GOPS computed by each process is known (see Figs. 6, 8, 10, 12 and 14). In this section, we propose asynchronous algorithms based on the previously presented synchronous algorithms, in which the next GOP to be computed by one process depends on the encoding state at the time of finishing the computation of the actual GOP. The main goals of the asynchronous algorithms are, on the one hand, to be able to balance the computing load, and, on the other hand, to reduce the parallel overhead introduced by the synchronization processes. As the computational load assigned to each parallel process depends on several factors, like the video content (i.e motion estimation complexity of one GOP depends on its motion activity), the number of frames to encode and the number of parallel processes. As the load balancing cannot be achieved using asynchronous versions of S-II and S-III (see Figs. 7–10), we have developed the asynchronous algorithms A-I, A-IV and A-V which are versions of S-I, S-IV, and S-V, respectively.

- Option A-I (LB, LP and RA): All processes encode the first I-frame and include it in the reference picture list. Then each process encodes the next uncoded GOP of the input video sequence. The GOP allocation order will depend on the order in which each process finishes the encoding its current GOP.
- Option A-IV (AI): As in Option A-I, each new GOP computation is assigned to the first process that becomes idle, but here all GOPs are composed of a single I-frame.
- Option A-V (LB, LP and RA): As in Option A-I, all processes encode the first I-frame and, after that, each process encodes a fixed number of contiguous GOPs (GOP_BLOCK). The new GOP_BLOCK allocation depends on which process becomes idle first.

Fig. 23 shows the parallel distribution performed when Option A-I is used. There is no synchronization process until the end of the overall computation. After the encoding process of the first GOP, i.e., one I-frame, each process encodes the first GOP regardless the previously encoded GOP. As said before, the parallel distribution of Option A-IV, shown in Fig. 25, is similar to Option A-I parallel distribution, taking into account that each GOP is composed by only one I-frame. One GOP computed by one process is denoted by $G_{(a\,b)}$ and $I_{G_{(a\ b)}}$ in Figs. 23 and 25, respectively, being a the rank of the parallel process and b the number of GOP computed by the process a. Also, on both figures, “x”, “y” and “z” are the total number of GOPs computed by process “0”, “1” and “N” respectively. Contrary to the synchronous algorithms, the number of GOPs computed by each process may vary depending on the computational load assigned to each process.

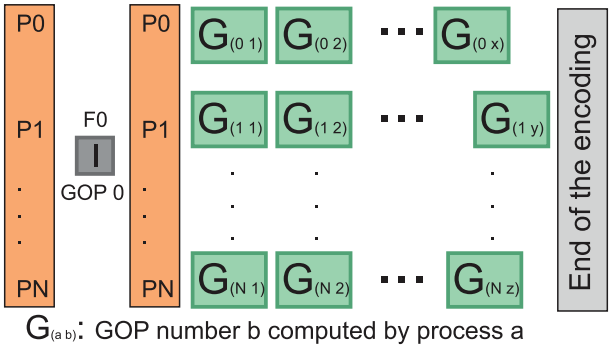


Fig. 23. Option A-I: parallel distribution.

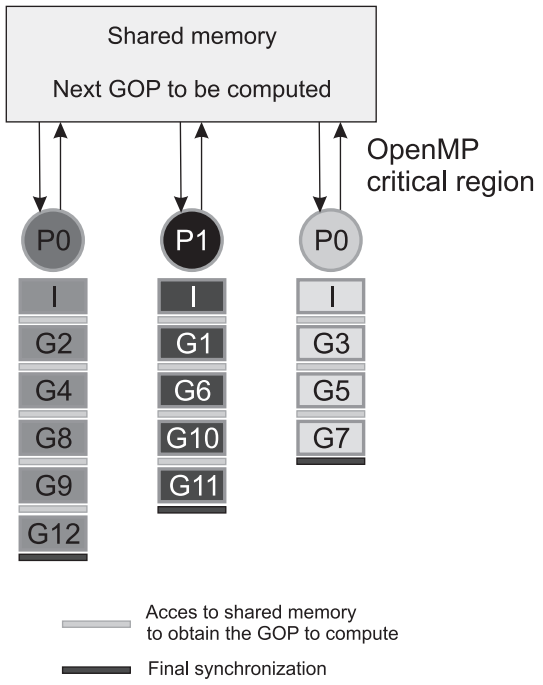


Fig. 24. Option A-I: example of parallel distribution.

In Fig. 24 we present an example for Option A-I, encoding 49 frames (the first I-frame and 12 GOPs of 4 frames). Note that we cannot ensure which GOPs are computed by each parallel process. After the first I-frame is encoded, each parallel process accesses to the shared memory in order to know the next GOP to be computed. This memory access is performed within an OpenMP critical region, in order to ensure that no more than one parallel process accesses to memory concurrently.

In Fig. 26 we present an example for Option S-IV in which each parallel process encodes the same number of GOPs. Therefore, as the number of frames to compute is 12, each parallel process encodes 4 frames. However, again, we cannot guarantee which GOPs are computed by each parallel process.

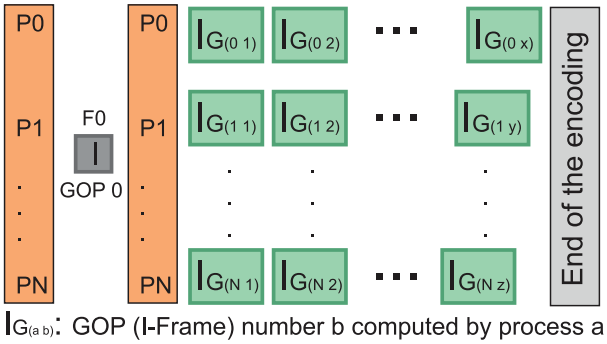


Fig. 25. Option A-IV: parallel distribution.

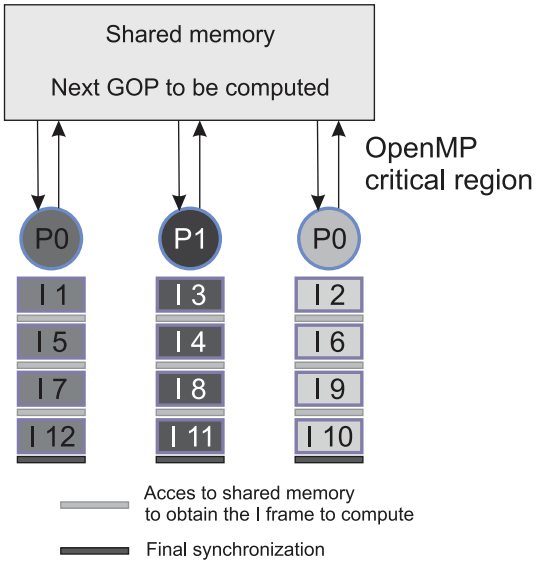


Fig. 26. Option A-IV: example of parallel distribution.

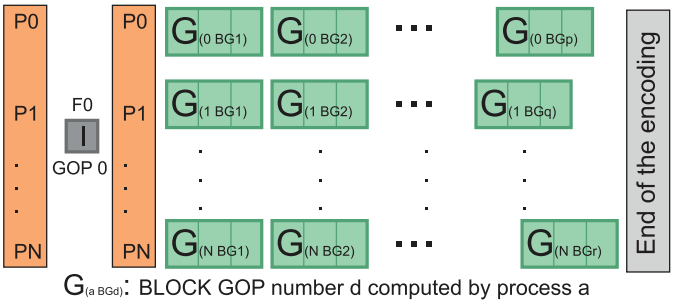


Fig. 27. Option A-V: parallel distribution.

Option A-V parallel distribution is shown in Fig. 27. As previously said in Section 3, the main goals of Option S-V are, (a) all processes are able to write data into the bit stream more frequently than Options S-II and S-III, and (b) to reduce PSNR and bit rate differences with respect to the sequential algorithm. In Option A-V the same two goals are pursued but due to the fact that the pattern of GOPs assigned to each process is determined at runtime, the level of achievement of the goals may vary from one execution to another. In Fig. 27 one GOP_BLOCK computed by one process is denoted by $G_{(a\,BGd)}$, where a denotes the rank of the parallel process and BGd denotes the GOP_BLOCK number d computed by the process a.

As we increase the GOP_BLOCK size, the asynchronous computation will probably be more similar to the synchronous model.

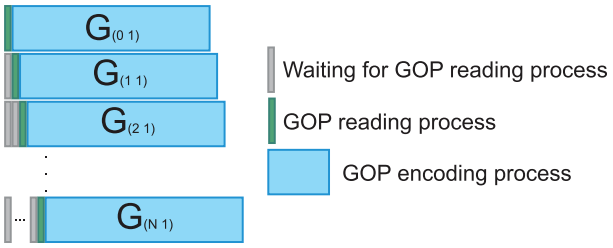


Fig. 28. Reading process.

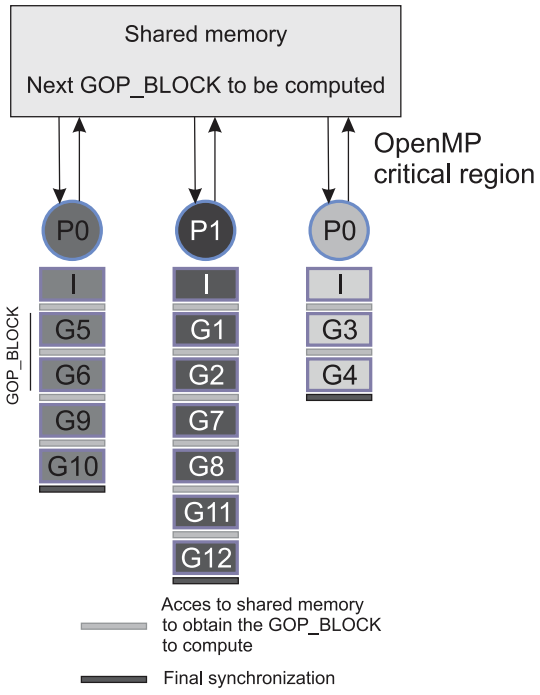
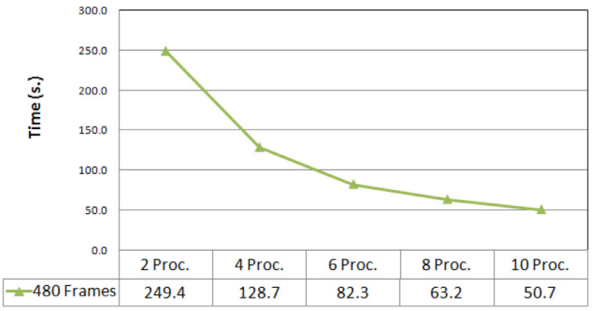


Fig. 29. Option A-V: example of parallel distribution.

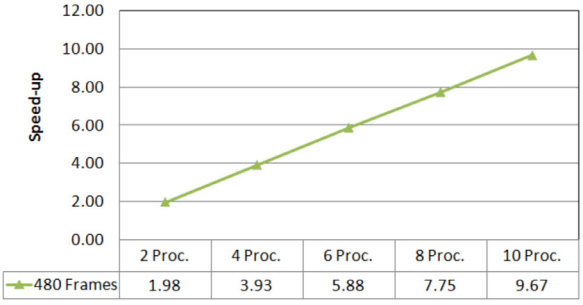
The asynchronous execution differs with respect to the synchronous execution when (due to the unbalanced computational load) one process completes its GOP (or GOP_BLOCK) computation before another lower rank process. As every encoding process begins with the reading of one complete GOP from the video sequence file, and reading is a sequential process, the encoding of the first GOP is not a simultaneous process. There is a small delay here, which accumulates as the parallel rank increases. Fig. 28 illustrates this behavior.

Note that in all asynchronous algorithms we cannot previously know, the frames to be computed by each parallel process. Fig. 29 shows an example for Option A-V encoding 49 frames. In contrast with Fig. 14, in which the execution is accurately described, the example shown in Fig. 29 is a simulation.

We want to remark that, in the asynchronous algorithms, a coordination process is needed in such a way that each parallel process could get, after the computation of a GOP, the next GOP to be encoded. As it can be seen in Figs. 24, 26 and 29, we use the shared memory to perform the required coordination process. In particular, we store in the global memory a variable containing the next GOP to be encoded. The first process that becomes idle reads and updates this variable. These reading and updating memory processes are performed inside a OpenMP critical region, avoiding problems due to concurrent accesses. As future work, in order to use distributed memory platforms, we will perform the



(a) Time (s.)



(b) Speed-up

Fig. 30. Option A-IV parallel algorithms when computing 480 frames.

Table 3
Computational times for the asynchronous parallel algorithms for the LB mode when computing 480 frames.

Algorithm	1 p	2 p	4 p	6 p	8 p	10 p
A-I	1807.2	1046.5	593.6	417.0	325.2	262.5
A-V	1807.2	1046.4	593.1	404.6	307.3	249.9

coordination task through messages, using the Message Passing Interface (MPI) [19].

4.1. Asynchronous algorithms evaluation

In this subsection we analyze the proposed asynchronous parallel algorithms in terms of parallel performance under the same conditions than the ones used in the synchronous algorithms evaluation.

Fig. 30 (a) shows the computational times for Option A-IV when encoding 480 frames while Fig. 30(b) shows the corresponding speed-ups. Option A-IV is related to the AI coding mode in which there are no dependencies between frames. This parallel algorithm offers good time reductions when increasing the number of processes, achieving efficiencies close to the ideal ones.

In Table 3 we present the computational times for Option A-I and Option A-V parallel algorithms using the LB encoding mode. Note that when using just 1 process, all proposed algorithms show similar timings than the ones obtained by the sequential version. Fig. 31 shows the speed-ups associated to the results shown in Table 3. This figure confirms a good behavior of the proposed parallel algorithms, especially for Option A-IV which obtains slightly better speed-ups than Option A-V.

5. Comparison results between synchronous and asynchronous algorithms

Finally, in this section we will compare the synchronous algorithms explained in Section 3, with respect to the asynchronous algorithms presented in Section 4.

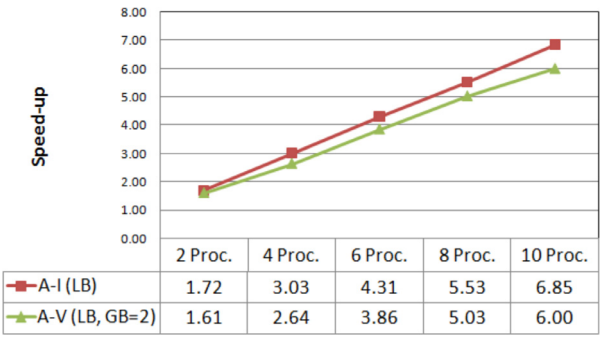


Fig. 31. Speed-up for the asynchronous parallel algorithms for the LB mode when computing 480 frames.

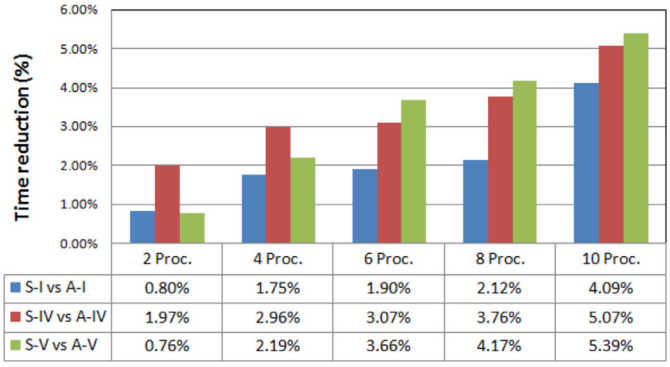
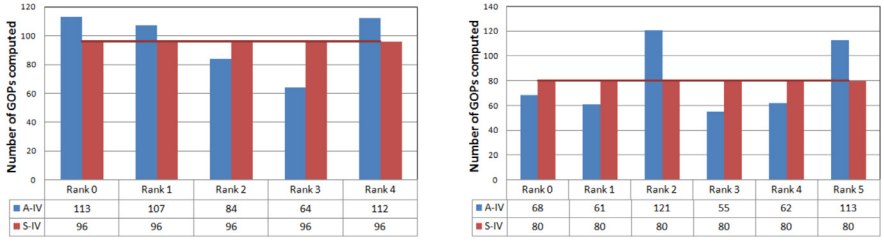


Fig. 32. Synchronous and asynchronous comparison.

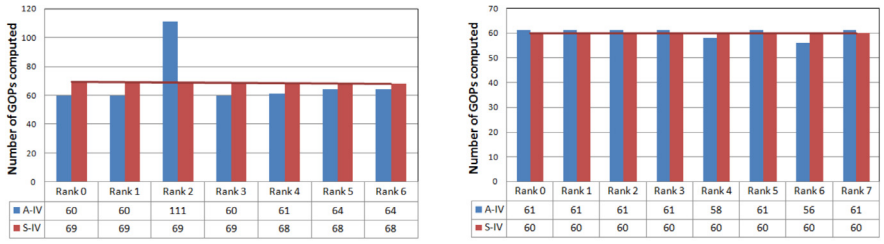
Fig. 32 shows a comparison, in terms of relative computational time savings, between synchronous and asynchronous algorithms. As it can be seen, asynchronous algorithms obtain a slight time reduction that is increased as the number of processes does. Since our parallel test platform is an homogeneous computing platform and the computational load is balanced, the reduction time is mainly due to the removal of the synchronization processes. Regarding the GOPs computed by each parallel process, we can confirm that all processes compute the same number of GOPs, i.e., the load is balanced.

As mentioned before, when using the asynchronous algorithms, the pattern of the GOPs computed by each process may vary depending on a particular execution, so both the bit rate and PSNR may be different depending on the order in which GOPs are allocated to each process. However, in our experiments the pattern of GOPs computed by each process remains unchanged respect to the one found with synchronous algorithms. Therefore, concerning to bit rate and PSNR, we can extend the conclusions of the synchronous algorithms to the asynchronous counterparts. We can consider the asynchronous algorithms significantly better when the load is unbalanced, either because the computing platform will be an heterogeneous computing platform, or because the power computing of some processors may be affected by other external processes, or because the GOPs assigned to each process involves divergent computational loads.

In order to verify this statement, we have used a computing platform, with only one quad core Intel Xeon F5640. As the GOP size for the AI mode is the smallest one (equal to 1), we have compared the synchronous S-IV mode with the asynchronous A-IV mode using up to 8 processes over 4 cores. In this case, as the number of processes is greater than the number of processors (cores), the computational load is unbalanced because one processor (core) must execute more than one process. In Fig. 33, we show the number of GOPs computed by each parallel process using a

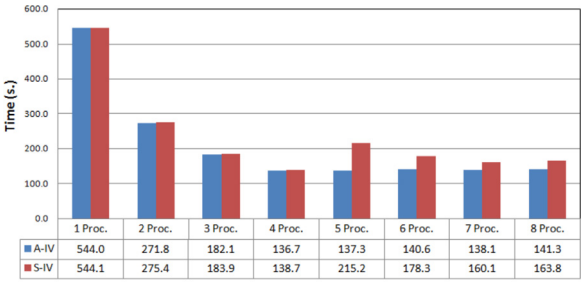


(a) 5 Processes. (b) 6 Processes.

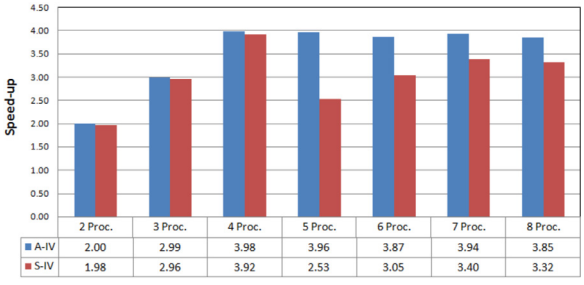


(c) 7 Processes. (d) 8 Processes.

Fig. 33. GOPs computed by each parallel process for the AI mode when computing 480 frames. QUADCORE platform.



(a) Time (s.).



(b) Speed-up.

Fig. 34. Synchronous and asynchronous comparison for the AI mode when computing 480 frames. QUADCORE platform.

larger number of processes than the available processors in the quad core platform. Obviously, when we use 5 processes one core must execute 2 processes, using 6 processes two cores must execute 2 processes and using 7 processes three cores must execute 2 processes while the remaining ones must execute just 1 process. Finally, when using 8 processes, all cores execute 2 processes, being again the load balanced. Looking at Fig. 33, we can confirm that the asynchronous mode automatically balances the load by assigning higher number of GOPs to processors running only one process.

Fig. 34 shows the computational times and speed-up using up to 8 processes in the quad core platform. This figure confirms that the computational results for the asynchronous mode are always better than for the synchronous mode, due to the load balancing performed in the asynchronous mode. Moreover, in the synchronous mode, the speed-up gets worse as the computational load is more unbalanced, i.e., for 5 processes in our experiment. However, in the asynchronous mode, the speed-up remains constant when the number of processes is higher than the number of cores, i.e., when the computational load is unbalanced.

6. Conclusions

In this paper we have proposed several parallel algorithms of the HEVC video encoder. These algorithms are based on a coarser grain parallelization approach with the organization of video frames in Group Of Pictures (GOP) and different GOP allocation schemes. A good parallel behavior has been shown in the experiments reported, which were obtained using a multicore platform. However the developed algorithms are able to run on distributed memory architectures since a coarser grain parallelization has been used. We have presented results using the different encoding modes proposed by the reference software, analyzing its performance. After implementing the algorithms in the HEVC software some experiments were performed showing interesting results as (a) GOP organization determines the final coding performance, being the best approach Option S-IV (AI mode) algorithm when comparing both sequential and parallel versions in terms of speed-up/efficiency; (b) although Option S-III algorithm introduces a bit rate overhead as the number of processes increases, the overall parallel performance and the improvements in PSNR make it a good approach when LB, LP or RA coding modes are demanded; (c) Option S-V algorithm offers similar features than Option S-III but with the ability to update the bit stream during encoding process with data obtained from all processes, not just from the root process; and (d) asynchronous versions of S-I, S-IV y S-V algorithms were provided showing slightly lower encoding times with the ability of load-balancing the input workload among the available processes. Some experiments prove this special feature that may be crucial in certain heterogeneous computing platforms or when source video has high variable motion content that produces irregular workloads at the encoder.

In general, all proposed versions attain high parallel efficiency results, showing that GOP-based parallelization approaches should be taken into account to reduce the HEVC video encoding complexity. As future work, we will explore hierarchical parallelization approaches combining GOP-based approaches with slice and tile

parallelization levels, which are aimed to exploit the shared memory parallelism rather than the distributed memory parallelism.

Acknowledgments

This research was supported by the Spanish Ministry of Education and Science under Grant TIN2011-27543-C03-03, the Spanish Ministry of Science and Innovation under Grants TIN2015-66972-C5-4-R and TIN2011-15734-E.

References

[1] Bross B., Han W., Ohm J., Sullivan G., Wang Y.-K., Wiegand T.. High efficiency video coding (HEVC) text specification draft 10. Document JCTVC-L1003. Geneva: JCT-VC.
[2] ITU-T, ISO/IEC JTC 1, Advanced video coding for generic audiovisual services, ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) version 16. 2012.
[3] Sullivan G, Ohm J, Han W, Wiegand T. Overview of the high efficiency video coding (HEVC) standard. IEEE Trans Circuits Syst Video Technol 2012;22(12):1648–67.
[4] Bossen F, Bross B, Suhring K, Flynn D. HEVC complexity and implementation analysis. IEEE Trans Circuits Syst Video Technol 2012;22(12):1685–96.
[5] Alvarez-Mesa M, Chi C, Juurlink B, George V, Schierl T. Parallel video decoding in the emerging HEVC standard. In: Proceedings of international conference on acoustics, speech, and signal processing, Kyoto; 2012. p. 1–17.
[6] Ayele E, Dhok SB. Review of proposed high efficiency video coding (HEVC) standard. Int J Comput Appl 2012;59(15):1–9.
[7] Chi CC, Alvarez-Mesa M, Juurlink B, Clare G, Henry F, Pateux S, et al. Parallel scalability and efficiency of HEVC parallelization approaches. IEEE Trans Circuits Syst Video Technol 2012;22(12):1827–38.
[8] Chi CC, Alvarez-Mesa M, Lucas J, Juurlink B, Schierl T. Parallel HEVC decoding on multi- and many-core architectures. J Signal Process Syst 2013;71(3):247–60.
[9] Bross B, Han W-J, Ohm J-R, Sullivan GJ, Wang Y-K, Wiegand T. High efficiency video coding (HEVC) text specification draft 10. Technical report JCTVC-l1003. Geneva, Switzerland: Joint Collaborative Team on Video Coding (JCT-VC); January 2013.
[10] Yu Q, Zhao L, Ma S. Parallel AMVP candidate list construction for HEVC. In: Proceedings of VCIP'12; 2012. p. 1–6.
[11] González D, Botella G, García C, Prieto M, Tirado F. Acceleration of block-matching algorithms using a custom instruction-based paradigm on a NIOS II microprocessor. EURASIP J Adv Signal Process 2013;1:1–20.
[12] González D, Botella G, Meyer-Baese U, García C, Sanz C, Prieto M, et al. A low cost matching motion estimation sensor based on the NIOS II microprocessor. Sensors 2012;12(10):13126–49.
[13] Jiang J, Guo B, Mo W, Fan K. Block-based parallel intra prediction scheme for HEVC. J Multimed 2012;7(4):289–94.
[14] Bolc L, Tadeusiewicz R, Chmielewski L, Wojciechowski K. Diamond scanning order of image blocks for massively parallel HEVC compression. Lect Notes in Comput Sci 2012;7594:172–9.
[15] Yan C, Zhang Y, Dai F, Liang L. Efficient parallel framework for HEVC deblocking filter on many-core platform. In: Proceedings of data compression conference (DCC); 2013.
[16] Migallón H, Hernández-Losada J, Cebrián-Márquez G, nol PP, Martínez J, López-Granado O. OpenMP HEVC parallel version based on a gop approach. Proceedings of the ninth international conference on engineering computational technology. Iványi P, Topping B, editors. Stirlingshire, UK: Civil-Comp Press; 2014. doi:10.4203/ccp.105.24.
[17] HEVC Reference Software, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-10.0/ [accessed 01.31.13].
[18] OpenMP Architecture Review Board. OpenMP application program interface, version 3.1. http://www.openmp.org [accessed 09.05.11].
[19] MPI Forum. MPI: a message-passing interface standard. Version 2.2. http://www.mpi-forum.org; December 2009 [accessed 04.09.09].